ONNX Runtime-ZenDNN User Guide

Publication # **57302**	Revision # **4.0**
Issue Date **January 2023**	

Contents

List of Figures

List of Tables

Revision History

Date	Revision	Description
January 2023	4.0	• Updated supported ONNX Runtime versions. • Added sections 1.1.1 and 7.3 through 7.8.
June 2022	3.3.1	Updated supported ONNX Runtime versions.
December 2021	3.2	• Updated supported ONNX Runtime version. • Added a new section “Optimal Settings” in Chapter 6. • Removed Chapter 7 (Blocked Format Support).
August 2021	3.1	Updated supported ONNX Runtime version.
April 2021	3.0	Initial version.

Chapter 1 Installing ZenDNN with ONNX Runtime

Note: *Refer to the ZenDNN v4.0 User Guide before starting the installation.*

1.1 Binary Release Setup

1.1.1 Conda

Complete the following steps to setup Conda:

1. Refer to Anaconda documentation (*https://docs.anaconda.com/anaconda/install/linux/*) to install Anaconda on your system.
2. Create and activate a Conda environment which will house all the ONNX Runtime-ZenDNN specific installations:

   ```
   conda create -n onnxrt-v1.12.1-ZenDNN-4.0-rel-env python=3.8 -y

   conda activate onnxrt-v1.12.1-ZenDNN-4.0-rel-env
   ```

   Ensure that you install the ONNX Runtime-ZenDNN package corresponding to the Python version with which you created the Conda environment.

   Note: *If there is any conda environment named onnxrt-1.12.1-zendnn-v4.0-rel-env already present, delete the conda environment onnxrt-1.12.1-zendnn-v4.0-rel-env (using command conda remove --name onnxrt-1.12.1-zendnn-v4.0-rel-env --all) before running scripts/ONNXRT_ZenDNN_setup_release.sh.*

3. It is recommended to use the naming convention:

   ```
   onnxrt-v1.12.1-ZenDNN-v4.0-rel-env
   ```

4. Install all the necessary dependencies:

   ```
   pip install -U cmake numpy==1.23.2 pytest psutil torch==1.13.0 coloredlogs

   pip install -U transformers sympy --ignore-installed ruamel.yaml

   pip install protobuf==3.20.1

   pip install onnx==1.12.0
   ```

   Note: *For binary packages built with Python v3.7, it is recommended to use numpy v1.21.6 (numpy==1.21.6).*

1.1.2 ONNX Runtime v1.12.1

Complete the following steps to install the ZenDNN binary release:

1. Copy the zipped release package to the local system being used. The name of the release packages will be similar to *ONNXRT_v1.12.1_ZenDNN_v4.0_Python_v*/.*
2. Execute the following commands:
   a. `unzip ONNXRT_v1.12.1_ZenDNN_v4.0_Python_v*.zip`
   b. `cd onnxrt-v1.12.1-ZenDNN-4.0-Python_v*/`
   c. `source scripts/ONNXRT_ZenDNN_setup_release.sh`

   Notes:

   1. *Ensure that it is sourced only from the unzipped release folder.*
   2. *After the installations, if there are warnings as follows, re-install the compatible protobuf from pip:*
      "ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source of the following dependency conflicts. onnx 1.12.0 requires ***protobuf<=3.20.1,>=3.12.2****, but you have protobuf 4.21.11 which is incompatible."*

The Python release binaries are tested with the recent Linux distributions such as:

- Ubuntu 20.04 and later
- RHEL 9.0 and later

Chapter 2 Directory Structure

The release folder consists of a ONNX Runtime wheel (.whl), LICENSE and THIRD-PARTY-PROGRAMS files, and the following directories:

- *scripts* contains scripts to set up the environment

Chapter 3 High-level Overview

The following is a high-level block diagram for the ZenDNN library, which uses the AOCL-BLIS library internally:

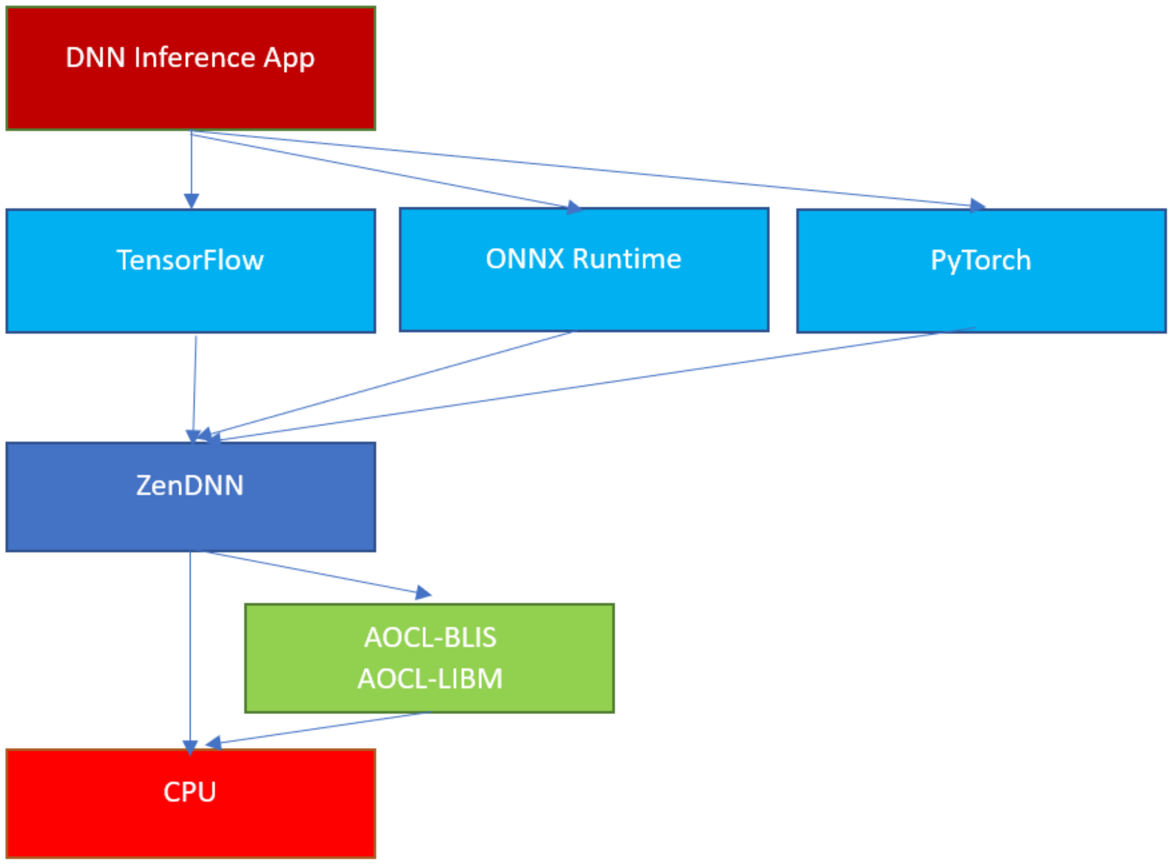

Figure 1. ZenDNN Library

In the current release, ZenDNN is integrated with TensorFlow, PyTorch, and ONNX Runtime.

Chapter 4 ONNX Runtime Benchmarks

To understand latency and throughput metrics with ZenDNN execution paths, standard models such as BERT can be run on ONNX Runtime with ZenDNN backend.

Use the following setup scripts and commands to download and run BERT models and derive performance metrics:

1. `cd ONNXRT_v1.12.1_ZenDNN_v4.0_Python_v*/`
2. `source scripts/zendnn_ONNXRT_env_setup.sh`
3. Activate your conda environment.
4. The following command runs BERT benchmarking with batch-size=4, sequence length=16, and threads=64 on a NPS=4 machine setting:

```
numactl --cpunodebind=0-3 --interleave=0-3 python -m onnxruntime.transformers.bench-
mark -m bert-large-uncased --model_class AutoModel -p fp32 -i 3 -t 10 -b 4 -s 16 -n 64
-v
```

Notes:

1. *For optimal settings, refer to the Tuning Guidelines section. Current setting refers to 96C, 2P, SMT=ON configuration.*
2. *If the following warning is displayed on the terminal, it can be ignored. During the environment setup, there is an optional script to gather information about hardware, OS, Kernel, and BIOS and it requires a few utilities (lscpu, lstopo-no-graphics, dmidecode, and*

so on) to be present. If these utilities are not present, you may encounter the following warning:

```
scripts/gather_hw_os_kernel_bios_info.sh

bash: lstopo-no-graphics: command not found

bash: lstopo-no-graphics: command not found

bash: lstopo-no-graphics: command not found

bash: lstopo-no-graphics: command not found

bash: lstopo-no-graphics: command not found

bash: lstopo-no-graphics: command not found

bash: lstopo-no-graphics: command not found

bash: _HW_LSTOPO_NUM_L2CACHE/_HW_LSTOPO_PACKAGES: division by 0 (error token is
"_HW_LSTOPO_PACKAGES")

bash: _HW_LSTOPO_NUM_L3CACHE/_HW_LSTOPO_PACKAGES: division by 0 (error token is
"_HW_LSTOPO_PACKAGES")

sudo: dmidecode: command not found

sudo: dmidecode: command not found

sudo: dmidecode: command not found
```

Chapter 5 ONNX Runtime v1.12.1

In this release of ZenDNN:

- ZenDNN library is supported for ONNX Runtime v1.12.1.
- AMD Unified Inference Frontend (UIF) optimized models are supported. For the model details, refer to the AMD UIF documentation.
- ONNX Runtime v1.12.1 wheel file is compiled with GCC v9.3.1.

Chapter 6 Environment Variables

ZenDNN uses the following environment variables to setup paths and control logs:

Table 1. ZenDNN Environment Variables-Generic

Environment Variable	Default Value/User Defined Value
ZENDNN_LOG_OPTS	ALL: 0
ZENDNN_PARENT_FOLDER	Path to unzipped release folder
ZENDNN_PRIMITIVE_CACHE_CAPACITY	The default value is set to 1024, you can modify it as required[a]
OMP_DYNAMIC	FALSE

a. These environment variables work only for Blocked Format.

The following is a list of environment variables to tune performance:

Table 2. ZenDNN Environment Variables-Optimization

Environment Variable	Default Value/User Defined Value
OMP_NUM_THREADS	The default value is set to 96. You can set it as per the number of cores in the user system[a].
OMP_WAIT_POLICY	ACTIVE
OMP_PROC_BIND	FALSE
GOMP_CPU_AFFINITY	Set it as per the number of cores in the system being used. For example, use 0-95 for 96-core servers.
ZENDNN_CONV_ADD_FUSION_ENABLE	The flag is to enable convolution and add operator fusion. It is disabled (set to 0) by default. You can modify it to 1 to enable the fusion. It is used to optimize executions on all the variants of ResNet models.
ZENDNN_RESNET_STRIDES_OPT1_ENABLE	The flag is to enable strides trick optimization for ResNet blocks. It is disabled (set to 0) by default. You can modify it to 1 to enable the optimization. It is used to optimize executions on all the variants of ResNet models.
ZENDNN_BN_RELU_FUSION_ENABLE	This flag is disabled by default. You can use export command in Linux to set it to 1 and enable it. It is used to optimize executions on limited CNN models.
ZENDNN_CONV_CLIP_FUSION_ENABLE	This flag is disabled by default. You can use export command in Linux to set it to 1 and enable it. It is used to optimize executions on all the variants of MobileNet models.
ZENDNN_CONV_RELU_FUSION_ENABLE	The flag is to enable convolution and relu operator fusion. It is enabled (set to 1) by default. You can modify it to 0 to disable the fusion. It is used to optimize executions on all the variants of ResNet models.
ZENDNN_CONV_ELU_FUSION_ENABLE	The flag is to enable convolution and elu operator fusion. It is disabled (set to 0) by default. You can modify it to 1 to enable the fusion. It is used to optimize executions on limited CNN models.
ORT_ZENDNN_ENABLE_INPLACE_CONCAT	This flag is used to perform in place concatenation of intermediate tensors arrays. It is disabled (set to 0) by default. You can modify it to 1 to enable the optimization. This optimization is useful for the variants of Inception and GoogleNet models.

Table 2. ZenDNN Environment Variables-Optimization

Environment Variable	Default Value/User Defined Value
ZENDNN_GEMM_ALGO	The default value is 3. You can modify it to one of the following: • 0 = Auto • 1 = BLIS path • 2 = partial-BLIS • 3 = ZenDNN jit path • 4 = ZenDNN partial jit path ***Note: Auto is an experimental feature.***

Note: *There are a few other environment variables that are initialized by the setup script, however these are not applicable for the binary release setup.*

When source *scripts/zendnn_ONNXRT_env_setup.sh* is invoked, the script initializes all the environment variables except the one(s) which must be set manually. The environment variable **ZENDNN_PARENT_FOLDER** is initialized relative to the path defined by the unzipped release folder. To ensure that the paths are initialized correctly, it is important that the script is invoked from the unzipped release folder.

Chapter 7 Tuning Guidelines

The hardware configuration, OS, Kernel, and BIOS settings play an important role in performance. The details for the environment variables used on a 4th Gen AMD EPYC™ server to get the best performance numbers are as follows:

7.1 System

A system with the following specifications has been used:

Table 3. System Specification

Model name	4th Gen AMD EPYC™ 9654P 96-Core Processor
CPU MHz	Up to 3.7 GHz
No of Cores	96
1P/2P	1
SMT: Thread(s) per Core	2
Mem-Dims	12x64 GB

7.2 Environment Variables

The following environment variables have been used:

ZENDNN_LOG_OPTS=ALL:0

OMP_NUM_THREADS=96

OMP_WAIT_POLICY=ACTIVE

OMP_PROC_BIND=FALSE

OMP_DYNAMIC=FALSE

ZENDNN_GEMM_ALGO=3

ZENDNN_PARENT_FOLDER=/home/<user_id>/my_work

ZENDNN_PRIMITIVE_CACHE_CAPACITY=1024

ZENDNN_ONNXRT_VERSION=1.12.1

ZENDNN_ONNX_VERSION=1.12.0

ZENDNN_CONV_ADD_FUSION_ENABLE=0

ZENDNN_RESNET_STRIDES_OPT1_ENABLE=0

GOMP_CPU_AFFINITY=0-95

ZENDNN_CONV_CLIP_FUSION_ENABLE=0

ZENDNN_BN_RELU_FUSION_ENABLE=0

ZENDNN_CONV_ELU_FUSION_ENABLE=0

ORT_ZENDNN_ENABLE_INPLACE_CONCAT=0

As mentioned in “Environment Variables” on page 14, the script *scripts/ zendnn_ONNXRT_env_setup.sh*, initializes all the environment variables except the one(s) which you must set manually. The environment variables **OMP_NUM_THREADS**, **OMP_WAIT_POLICY**, **OMP_PROC_BIND**, and **GOMP_CPU_AFFINITY** can be used to tune performance. For optimal performance, the **Batch Size** must be a multiple of the total number of cores (used by the threads). On a 4th Gen AMD EPYC server (configuration: AMD EPYC 9654P 96-Core, 2P, and **SMT=ON**) with the above environment variable values, **OMP_NUM_THREADS=96** and **GOMP_CPU_AFFINITY=0-95** yield the best throughput numbers for a single socket.

Batch Size is a sensitive factor for the throughput performance of any model. The following formula could be used to calculate the optimal **Batch Size**:

Batch Size = number_of_physical_cores * batch_factor

batch_factor may vary from 8-32. Usually, the value 32 gives the optimal performance.

7.3 Thread Wait Policy

OMP_WAIT_POLICY environment variable provides options to the OpenMP runtime library based on the expected behavior of the waiting threads. It can take the abstract values **PASSIVE** and **ACTIVE**. The default value is **ACTIVE**. When **OMP_WAIT_POLICY** is set to **PASSIVE**, the waiting threads will be passive and will not consume the processor cycles. Whereas, setting it to **ACTIVE** will consume processor cycles.

Note: *For ZenDNN stack, setting **OMP_WAIT_POLICY** to **ACTIVE** may give better performance.*

7.4 Thread Affinity

To improve ZenDNN performance, the behavior of OpenMP thread can be guarded precisely with thread affinity settings. A thread affinity defined at start up cannot be modified or changed during runtime of the application. Following are the ways through which you can bind the requested OpenMP threads to the physical CPUs:

- **GOMP_CPU_AFFINITY** environment variable binds threads to the physical CPUs, for example:

  export GOMP_CPU_AFFINITY="0 3 1-2 4-15:2"

  This command will bind the:

  – Initial thread to CPU 0
  – Second thread to CPU 3
  – Third and fourth threads to CPU 1 and CPU 2 respectively
  – Fifth thread to CPU 4
  – Sixth through tenth threads to CPUs 6, 8, 10, 12, and 14 respectively

  Then, it will start the assigning back from the beginning of the list.

  export GOMP_CPU_AFFINITY="0" binds all the threads to CPU 0.

- Kmp affinity belongs to LLVM OpenMP runtime library and is used by setting appropriate values for the environment variable **KMP_AFFINITY**. It has the following syntax:

  KMP_AFFINITY=[<modifier>,...]<type>[,<permute>][,<offset>]

  Example:

  export KMP_AFFINITY='verbose,respect,granularity=fine,compact,1,0 binds threads as close as possible to the master thread but on a different core. Once each core is assigned with one OpenMP thread, the remaining OpenMP threads are assigned in the same order as before, but on different thread context.

  Among the two, **KMP_AFFINITY** takes highest precedence followed by **GOMP_CPU_AFFINITY**. If none of the two is set, the host system will defer the assignment of threads to CPUs. Given the same thread binding (see example below), it is expected that both the affinity settings would give the same performance.

  Example:

  Following affinity settings should give the same thread bindings:

  – **export GOMP_CPU_AFFINITY=0-95**
  – **export KMP_AFFINITY='verbose,respect,granularity=fine,compact,1,0'**

7.5 Non-uniform Memory Access

7.5.1 numactl

numactl provides options to run processes with specific scheduling and memory placement policy. It can restrict the memory binding and process scheduling to specific CPUs or NUMA nodes:

- `--cpunodebind=nodes`: Restricts the process to specific group of nodes.
- `--physcpubind=cpus`: Restricts the process to specific set of physical CPUs.

- `--membind=nodes`: Allocates the memory from the nodes listed. The allocation fails if there is not enough memory on the listed nodes.
- `--interleave=nodes`: Memory will be allocated in a round robin manner across the specified nodes. When the memory cannot be allocated on the current target node, it will fall back to the other nodes.

Example:

If <tensorflow_script> is the application that needs to run on the server, then it can be triggered using `numactl` settings as follows:

```
numactl --cpunodebind=0-3 -membind=0-3 python <tensorflow_script>
```

The `interleave` option of numactl works only when the number nodes allocated for a particular application is more than one. `cpunodebind` and `physcpubind` behave the same way for ZenDNN stack, whereas `interleave` memory allocation performs better than `membind`.

7.5.2 Concurrent Execution

As every application, AI workload requires special considerations during performance tuning to get the best out of the non-uniform memory access (NUMA) enabled machine. Improvement in performance can be achieved by carefully analyzing memory access time, memory bandwidth, and congestion on the shared bus. These factors depend on how far away the allocated memory and the process that requested the memory are in the NUMA system. In NUMA machines, the local memory access is faster as compared to the remote memory access. Consider the following workload:

```
numactl --cpunodebind=0-3 -membind=0-3 python <tensorflow_script>
```

Performance can be optimized by partitioning the workload into multiple data shards and then running concurrently on more than one NUMA node. Following example shows the concurrent execution across 4 NUMA nodes:

```
numactl --cpunodebind=0 --membind=0 python <tensorflow_script> & numactl --cpuno-
debind=1 --membind=1 python <tensorflow_script> & numactl --cpunodebind=2 --membind=2
python <tensorflow_script> & numactl --cpunodebind=3 --membind=3 python <tensor-
flow_script>
```

The number of concurrent executions can be increased beyond 4 nodes. The following formula can be used to decide the number of concurrent executions to be triggered at a time:

```
Number Concurrent Executions = Number of Cores Per Socket / Numbers of Cores sharing L3
cache
```

This can also be extended to even cores. However, these details should be verified by the user empirically.

7.6 Transparent Huge Pages

Transparent Huge Pages (THPs) are a Linux kernel feature for memory management to improve performance of the application by efficiently using processor's memory-mapping hardware. THP should reduce the overhead of the Translation Lookaside Buffer. User must login as a sudo user to enable or disable THP settings. It operates mainly in two modes:

- always: You can run the following command to set THP to ‘always’:

  ```
  echo always > /sys/kernel/mm/transparent_hugepage/enabled
  ```

  In this mode, the system kernel tries to assign huge pages to the processes running on the system.

- madvise: You can run the following command to set THP to ‘madvise’:

  ```
  echo madvise > /sys/kernel/mm/transparent_hugepage/enabled
  ```

  In this mode, kernel only assigns huge pages to the individual processes memory areas.

You can use the following command to disable THP:

```
echo never > /sys/kernel/mm/transparent_hugepage/enabled
```

It is recommended to use the following THP setting for better performance:

- CNN models - ‘always’
- NLP models - ‘madvise’

7.7 Batch Size

Batch Size is a sensitive factor for the throughput performance of any model. The following formula could be used to calculate the optimal Batch Size:

```
 Batch Size = number_of_physical_cores * batch_factor
```

Batch factor may vary from 8-32. The value 32 may provide optimal performance. However, user should verify this empirically.

7.8 Memory Allocators

Based on the model, if there is a requirement for a lot of dynamic memory allocations, a memory allocator can be selected from the available allocators which would generate the most optimal performance out of the model. These memory allocators override the system provided dynamic memory allocation routines and use a custom implementation. They also provide the flexibility to override the dynamic memory management specific tunable parameters (for example, logical page size, per thread, or per-cpu cache sizes) and environment variables. The default configuration of these allocators would work well in practice. However, you should verify empirically by trying out what setting works best for a particular model after analyzing the dynamic memory requirements for that model.

Most commonly used allocators are TCMalloc and JEMalloc.

7.8.1 TCMalloc

TCMalloc is a memory allocator which is fast, performs uncontended allocation and deallocation for most objects. Objects are cached depending on the mode, either per-thread or per-logical CPU. Most allocations do not need to take locks. So, there is low contention and good scaling for multi-threaded applications. It has flexible use of memory and hence, freed memory can be reused for different object sizes or returned to the operating system. Also, it provides a variety of user-accessible controls that can be tuned based on the memory requirements of the workload.

7.8.2 JEMalloc

JEMalloc is a memory allocator that emphasizes fragmentation avoidance and scalable concurrency support. It has a powerful multi-core/multi-thread allocation capability. The more cores the CPU has, the more program threads, the faster JEMalloc allocates. JEMalloc classifies memory allocation granularity better, leading to less lock contention. It provides various tunable runtime options, such as enabling background threads for unused memory purging, allowing JEMalloc to utilize transparent huge pages for its internal metadata, and so on.

7.8.3 Usage

You can install the TCMalloc/JEMalloc dynamic library and use **LD_PRELOAD** environment variable as follows:

```
Before using TCMalloc:
export LD_PRELOAD=/path/to/TCMallocLib/

Before using JEMalloc:
export LD_PRELOAD=/path/to/JEMallocLib/

Or

Benchmarking command using TCMalloc:
LD_PRELOAD=/path/to/TCMallocLib/ < python benchmarking command>

Benchmarking command using JEMalloc:
LD_PRELOAD=/path/to/JEMallocLib/ < python benchmarking command>
```

To verify if TCMalloc/JEMalloc memory allocator is in use, you can grep for tcmalloc/jemalloc in the output of `lsof` command:

```
lsof -p <pid_of_benchmarking_commad> | grep <tcmalloc/jemalloc>
```

7.9 Optimal Setting

Optimal performance of several ZenDNN workloads is observed when interleaving is enabled in conjunction with the NPS4 mode.

By default, ONNX Runtime uses GNU OpenMP (libgomp) for parallel computation. For ZenDNN backend, you can preload LLVM OpenMP's libomp for a better performance compared to libgomp as follows:

1. Download:

   ```
   cd $ZENDNN_PARENT_FOLDER
   ```

2. Unzip:

   ```
   tar -xf openmp-10.0.1.src.tar.xz
   cd openmp-10.0.1.src
   ```

3. Web get:

   ```
   wget https://github.com/llvm/llvm-project/releases/download/llvmorg-10.0.1/openmp-
   10.0.1.src.tar.xz
   ```

4. Configure cmake command:

   ```
   cmake -DCMAKE_C_COMPILER=gcc -DCMAKE_CXX_COMPILER=g++
   ```

5. Build command:

   ```
   make
   ```

6. Clean command:

   ```
   make clean
   ```

7. Build artifacts location:

   ```
   $ZENDNN_PARENT_FOLDER/openmp-10.0.1.src/runtime/src/libomp.so
   ```

8. Do LD_PRELOAD to the *.so* file:

   ```
   export LD_PRELOAD=$ZENDNN_PARENT_FOLDER/openmp-10.0.1.src/runtime/src/libomp.so:$LD_PRE-
   LOAD
   ```

This is a one-time activity and for all the benchmarking, you can just point to the already built lib.

A sample command line to run a Python code with 64C in NPS4 mode is as follows:

```
export GOMP_CPU_AFFINITY=0-63 && export OMP_NUM_THREADS=64 && numactl --cpunodebind=0-3 --
interleave=0-3 python -m onnxruntime.transformers.benchmark -m bert-large-uncased --mod-
el_class AutoModel -p fp32 -i 3 -t 10 -b 4 -s 16 -n 64 -v
```

Chapter 8 License

ZenDNN is licensed under Apache License Version 2.0. Refer to the “LICENSE” file for the full license text and copyright notice.

This distribution includes third party software governed by separate license terms.

3-clause BSD license:

- Xbyak (*https://github.com/herumi/xbyak*)
- Googletest (*https://github.com/google/googletest*)
- Instrumentation and Tracing Technology API (*https://github.com/intel/ittapi*)

Apache License Version 2.0:

- oneDNN (*https://github.com/oneapi-src/oneDNN*)
- Xbyak_aarch64 (*https://github.com/fujitsu/xbyak_aarch64*)

Boost Software License, Version 1.0:

Boost C++ Libraries (*https://www.boost.org/*)

MIT License from ONNXRT:

https://github.com/microsoft/onnxruntime

This third-party software, even if included with the distribution of the Advanced Micro Devices software, may be governed by separate license terms, including without limitation, third-party license terms, and open-source software license terms. These separate license terms govern use of the third-party programs as set forth in the THIRD-PARTY-PROGRAMS file.

Chapter 9 Technical Support

Please email *zendnnsupport@amd.com* for questions, issues, and feedback on ZenDNN.